Tool Support for Change-Based Regression Testing: An Industry Experience Report

Rudolf Ramler[1(✉)], Christian Salomon[1], Georg Buchgeher[1], and Michael Lusser[2]

[1] Software Competence Center Hagenberg, Softwarepark 21, 4232 Hagenberg, Austria
{rudolf.ramler, christian.salomon, georg.buchgeher}@scch.at

[2] OMICRON Electronics GmbH, Oberes Ried 1, 6833 Klaus, Austria
michael.lusser@omicron.at

Abstract. Changes may cause unexpected side effects and inconsistencies. Regression testing is the process of re-testing a software system after changes have been made to ensure that the new version of the system has retained the capabilities of the old version and that no new defects have been introduced. Regression testing is an essential activity, but it is also time-consuming and costly. Thus, regression testing should concentrate on those parts of the system that have been modified or which are affected by changes. Regression test selection has been proposed over three decades ago and, since then, it has been frequently in the focus of empirical studies. However, regression test selection is still not widely adopted in practice. Together with the test team of an industrial software company we have developed a tool-based approach that assists software testers in selecting regression test cases based on change information and test coverage data. This paper describes the main usage scenario of the approach, illustrates the implemented solution, and reports on its evaluation in a large industry project. The evaluation showed that the tool support reduces the time required for compiling regression test suites and fosters an accurate selection of regression test cases. The paper concludes with our lessons learned from implementing the tool support in a real-world setting.

Keywords: Regression testing · Test case selection · Change-based testing

1 Introduction

"Change is continual" is the first of the five laws by Lehman and Belady [1] that characterize the dynamics of program evolution. Changes to the software often cause unexpected side effects and inconsistencies with other parts of the software. Regression testing is the process of re-testing a software system after changes have been made to ensure that the new version of the system has retained the capabilities of the old version and that no new defects have been introduced [2]. While regression testing is an essential and valuable quality assurance measure, it is also a time-consuming and costly activity [3]. The required efforts and costs can be reduced by concentrating primarily on those tests, which exercise the parts of the system that have been changed or that are

D. Winkler et al. (Eds.): SWQD 2017, LNBIP 269, pp. 133–152, 2017.
DOI: 10.1007/978-3-319-49421-0_10

affected by changes. However, selecting a minimal relevant set of regression test cases is challenging as it requires in-depth knowledge about the system's implementation to spot all direct and indirect dependencies between the changes made in development and the related test cases.

The idea of regression test selection has been proposed over three decades ago [4–6]. Since then many different approaches for selecting test cases to compile an optimal regression test suite have been proposed [5, 6]. However, regression test selection has not been widely adopted in practice [7] and reports about successful applications in real-world projects are still rare [3, 8].

In this paper we describe an approach for change-based regression testing that has been implemented in a large industry project. The approach is supported by a tool named *Sherlock* that assists testers in selecting regression test cases based on changes made to the software system. The tool integrates information about test cases, changes and dependencies from different sources of the development lifecycle. The information is used to support testers in making decisions which test cases to select when compiling a regression test suite.

The industry context and motivation for our work are described in Sect. 2. Section 3 provides an overview of the approach for change-based regression testing and the implemented tool-support. The tool has been evaluated together with the company's test team. Section 4 describes the evaluation approach, and Sect. 5 discusses the results. The paper concludes with sharing our lessons learned and an outlook on future work in Sect. 6.

2 Industry Context and Motivation

This section describes the background of the studied company and the software system subject to testing. It illustrates the challenges experienced by the testers when selecting regression test cases in this context. Finally, it describes the derived goals and practical requirements for implementing tool support for regression testing.

2.1 Company Background and Software System

The tool support for regression test case selection has been developed together with test engineers of OMICRON electronics GmbH[1], an international software company in the electrical engineering domain. The company offers hardware and software products. One of these is a large software solution which has a development history of about 20 years. Over these years the system has grown tremendously in size as well as in complexity. The current version consists of over 2.5 million lines of code (MLOC) containing different programming languages and a mix of different software technologies. The system is structured into more than 40 functional modules. The modules interact with each other and share a common framework as well as various base libraries and hardware drivers. The system has grown to its current size due to

[1] https://www.omicronenergy.com/.

numerous contributions made by many different people in the role of developers, architects, product owners and testers. With the continuous growth of the system also the amount of dependencies between the different modules, libraries and layers has increased. Thus, today, one of the foremost challenges of effective and efficient regression testing lies in acquiring and managing the knowledge about the huge amount of dependencies in the software system.

The system is still under active development and new versions extending the rich set of features are released on a regular basis. We followed the development for about 1.5 years (20 months). In this time approximately 1.300 work items (features, change requests, and bug fixes) were implemented by 36 different people resulting in changes to 115.000 methods part of 3.600 source code files.

A team of five testers has to ensure that newly implemented functionality performs as expected and – by running regression tests – that these changes do not adversely affect the existing, unchanged functionality. For each of the functional modules the testers maintain up to several hundred test cases. Overall, more than 5,000 test cases exist for the whole system. The software system is primarily tested manually. Automated tests are only available for a small part of the system, mainly on the level of technical interfaces. Most test cases require manual interaction because the functional modules are user interface centered and, furthermore, they also have strong dependencies on hardware equipment that has to be setup and operated manually.

For a large, complex system like the one described in this paper, executing all available tests in regression testing is impossible due to time and resource constraints. Therefore, a relevant subset of all available test cases has to be selected. The problem, however, lies in knowing which test cases are relevant. The testers select regression tests based on the textual description of the implemented feature or fixed bug and their detailed knowledge of the system which they acquired over years of testing. For less experienced testers, e.g., those who are new to the team, it is particularly hard to select all relevant test cases due to the complexity of the system and due to the many dependencies that can only be traced on code level. In order to avoid missing relevant test cases, less experienced testers often tend to select unnecessarily large sets of regression test cases. These sets contain unrelated test cases that increase the testing effort but not the chance of spotting side-effects induced by the changes.

2.2 Challenges of Experience-Based Regression Testing

Figure 1 illustrates different situations a tester may face in regression testing after the developers have made changes to the system, e.g., due to a bug fix. In each of the six examples (*a* - *f*) three methods have been changed (shown as dark gray blocks). The ellipses show the coverage footprint of the available test cases (*test case A* to *test case G*), i.e., the methods covered when the test case is executed.

In the following examples *test case A* (shown as gray filled ellipse) represents the error-revealing test case that has initially been executed by the tester when the bug was found. This test case is usually linked to the bug report. It will be re-executed when the bug has been resolved by the developers to make sure the change works as expected

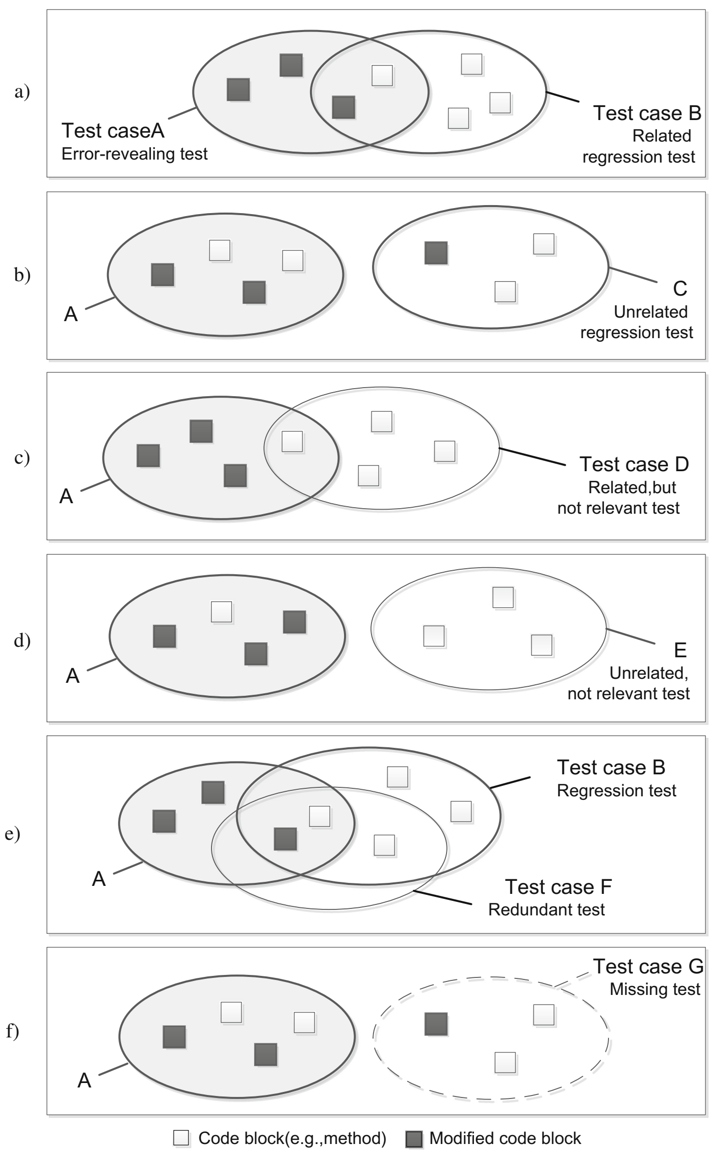


Fig. 1. Examples illustrating scenarios where test cases are selected for regression testing (bold ellipse), are not considered relevant (thin ellipse) or are missing (dashed ellipse).

and the bug report can be closed. In addition, the testers run regression tests to make sure the fix did not lead to unintended side effects.

The selection of regression test cases is based on the testers' personal knowledge and intuition. Different testers may therefore propose different sets of regression test cases. Generally the testers try to be as inclusive as possible, but they also try to keep the regression test set as small as possible to warrant efficiency. The examples illustrate the different situations the testers may face.

Example (a). The tester usually knows the usage scenario that initially revealed the bug. It is depicted as *test case A* (gray filled ellipse). In re-testing the tester runs this test case to make sure the bug has actually been fixed. One of the changed methods is also relevant for another usage scenario, shown as *test case B* (bold ellipse). Test case B has to be selected for regression testing. It makes sure that this alternative usage scenario still works in the same way as before and that it is not negatively affected by the change. Testers are usually able to identify suitable regression test cases when the usage scenarios of the initially error-revealing test and the regression test are overlapping or closely related, e.g., when they affect the same functional module.

Example (b). When changes are spread out across the system they often have "unexpected" side-effects occurring in different functional modules. This situation makes selecting relevant test cases (*test case C*) particularly hard. It requires detailed knowledge of the system which is often possessed only by experienced testers.

Example (c). Testing further, unrelated usage scenarios that do not depend on any of the changed methods such as *test case D* (thin ellipse) will not be able to spot side-effects and can therefore be omitted from regression testing. This is usually the case for most of the test cases. Usually all test cases relating to modules other than the one directly affected by the change can be excluded.

Example (d). However, test cases for usage scenarios similar to the error-revealing scenario are sometimes selected for regression testing, even when they do not cover any of the changed methods (*test case E*). Without knowledge of the changed code it can be very hard or even impossible for the testers to decide which of the test cases are relevant for regression testing and which are not.

Example (e). Sometimes test cases exercise almost the same usage scenarios but observe the system's behavior from a different viewpoint, e.g., functional correctness versus performance or results displayed on the user interface versus stored in the database. Similarly, larger test cases (*test case B*) sometimes subsume other tests (*test case F*), which focus on a small, specific aspect of the system. Such a relationship between test cases can usually be observed by comparing their coverage footprints. A tester may consider test cases that have the same or a smaller coverage footprint as redundant, depending on whether the tests' particular viewpoints or specialization is relevant in context of regression testing.

Example (f). Finally, despite the comprehensive set of test cases that exists in the test management system, it may still be possible that some of the changed methods are not covered by any of the existing test cases in the test management system. Thus, the tester has to come up with a new test case (*test case G*, shown as dashed ellipse) to be able to fully cover the change. Since testers are often not aware of the coverage footprints of the existing test cases, such uncovered changes may easily slip through regression testing. Thus, the testers are also performing exploratory testing in addition to running regression tests based on the test cases specified in the test management system.

2.3 Goals and Requirements

The goal of developing tool support for regression testing was to aid the testers in selecting a set of relevant test cases. Regression testing is considered to be safe [4] if all test cases are selected that may reveal a fault, i.e., a negative side-effect resulting from a change. Accordingly, all test cases may be selected that cover any of the changed parts of the system. However, the set of selected test cases should also be minimal ("small") in order to keep the required effort and time involved in regression testing as low as possible.

It is worth noting that the goal was not to cut testing costs or to reduce the length of testing cycles, but to increase the defect detection capability of regression testing. In the time-frame available for testing the testers should focus on the most relevant test cases covering the critical and the changed functionality to minimize risk of defects slipping through to production. Hence, on the long run, this approach is expected to reduce effort and costs by avoiding the usually expensive hotfixes and service release.

Following requirements were derived from these overall goals.

(1) Find all test cases that cover the changes. In order to reveal a fault, a test case has to be able to trigger the fault by executing the faulty part of the code. Thus, a test case should be selected for the initial set of regression tests if it fully or at least partially covers the changed code. Since changes were related to methods, the measure *method coverage* was proposed to select relevant test cases.

(2) Exclude unrelated test cases. Test cases that do not contribute to the coverage should be excluded from the initially proposed set of tests to keep the regression test suite small. Nevertheless, exceptions should be possible and testers should still be able to add "unrelated" test cases, for example, critical and high priority tests that have to be part of every regression test run.

(3) Identify redundant test cases. Changed parts may be covered by several similar test cases. Including all these test cases may lead to a highly redundant regression test suite. Testers should therefore be able to skip redundant test cases by deselecting them from the regression test suite.

(4) Interactive decision support. The tool should support human testers by automatically proposing a set of relevant test cases that are compiled into a regression test suite with minimal redundancy. However, the aim is not to replace the human tester and his knowledge, but to compile the relevant information, provide a comfortable overview, and support making quick yet sound decisions.

(5) Integration with existing tools. Regression testing relies on information that is maintained in tools currently used in testing and development. The regression tests are selected from the test cases stored in the company's test management system. Overall, it contains several thousand test cases. The test management system is also used to define and manage the test runs and to document test execution results. Additional tools exist for profiling and measuring test coverage. Change information has to be retrieved from the versioning system of the Microsoft Team Foundation Server used by the development team.

3 Approach and Tool Support

Numerous approaches for regression test case selection and prioritization have been investigated in empirical research. A comprehensive overview of the related work can be found in the literature reviews published by Engström, Runeson and Skoglund [5] and by Yoo and Harman [6]. An overview of code-based change impact analysis techniques has been provided by Li et al. [9]. The approach described in this paper assists testers in the selection of test cases related to changes made to the software system. Thus, it can be classified as test case selection approach, which deals with the selection of a subset of tests that check system modifications [6]. The approach has been implemented in form of the tool *Sherlock*. Its implementation is related to the graph-walk technique [10] and the modification-based technique [11]. Potentially relevant test cases are identified based on their coverage footprint, which is determined by recorded code coverage information from previous test runs. The coverage footprint provides the dependency information necessary to link test cases to the modified code.

In the following subsections we give an overview of Sherlock's integration with other tools and its high-level architecture, and we describe the steps how the tool support is used for compiling a regression test suite.

3.1 Data Sources and Tool Architecture

Figure 2 shows the overall architecture of the tool Sherlock – split into a client and a server part – and its interfaces to the testing tools serving as data sources.

Data Sources. For identifying and selecting the relevant regression test cases, Sherlock incorporates information from three data sources: (1) information about changes is retrieved from the *version control system* of Microsoft's Team Foundation Server (TFS)[2], (2) the list of available test cases and their properties are retrieved from the *test management system* SilkCentral Test Manager[3], and (3) information about which test cases are related to the source code changes is extracted from *code coverage analysis* results produced in previous test runs for the individual test case with the profiler SmartBear AQtime Pro[4].

Adapter. Dedicated adapters have been implemented for each of the three data sources. The adapters are used to extract, transform and load (ETL) the data into Sherlock's central data store. For data extraction the typically proprietary interfaces of the different tools are used. For example, Microsoft's TFS provides a REST API to retrieve a wide range of information including information about changed code in form of change sets attached to work items, the test management system's export interface is used to gain the list of available test cases, and the coverage information is extracted from coverage result files in XML format. These data sets are transformed into a tool-agnostic graph format consisting of nodes and edges. Nodes represent check-ins,

2 https://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx.

3 http://www.borland.com/en-GB/Products/Software-Testing/Test-Management/Silk-Central.

4 https://smartbear.com/product/aqtime-pro/overview/.

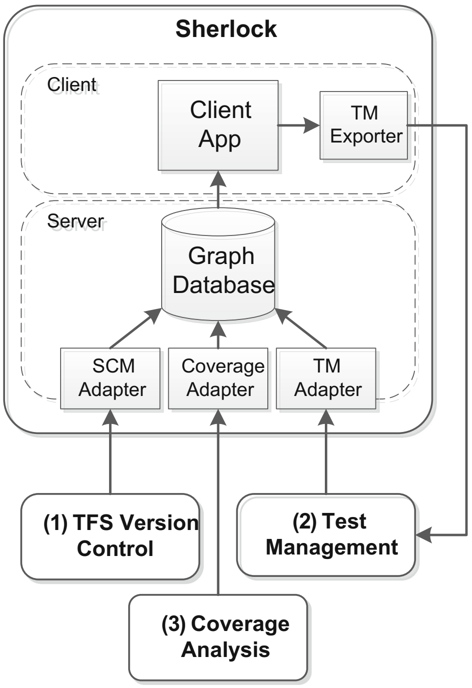


Fig. 2. High-level architecture of Sherlock and its interfaces to other test tools.

files, methods, test cases, etc. Edges represent dependencies such as those between check-ins and files, files and methods, as well as dependencies between methods and test cases. An example for one of the more complex transformations in the ETL process is computing the list of methods of a class that have been changed, based on the information of which lines in a source code file have been changed, added or removed. While the lines can be easily retrieved on file level from TFS, there is no support for linking this information to source code entities such as methods [12]. Finally, the nodes and edges are stored into a graph database.

Graph Database. Sherlock uses Neo4j[5], a NoSQL database optimized for storing and retrieving data structured in form of a graph. The database supports the creation of a simple graph-oriented data model and provides an intuitive query language. Different types of nodes can be defined, which can be combined by edges representing different types of relationships. Since optimized for scalability, the database is able to handle a huge number of nodes and dependencies. Figure 3 shows a basic data structure containing four different types of nodes and three types of relationships. Besides work items, change sets, methods and test cases, the database also contains nodes for source code files, source folders, branches and developers. The graph database runs as server and allows several testers working on regression test suits in parallel.

5 https://neo4j.com/.

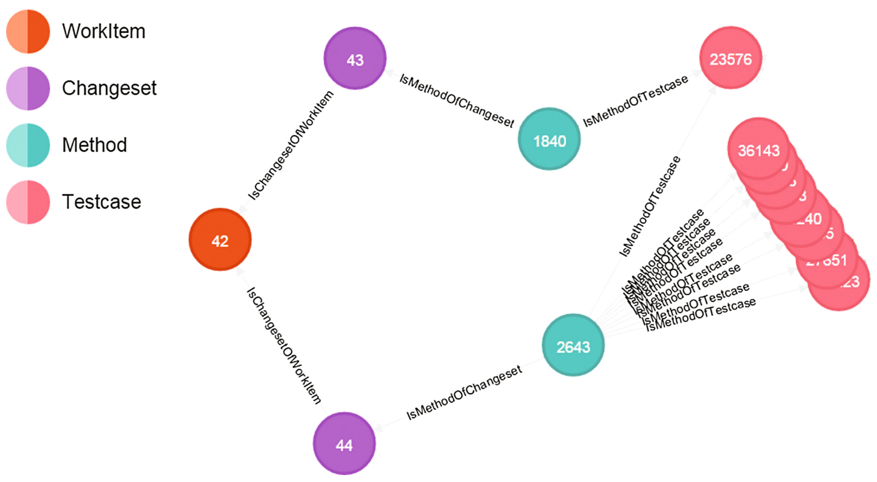


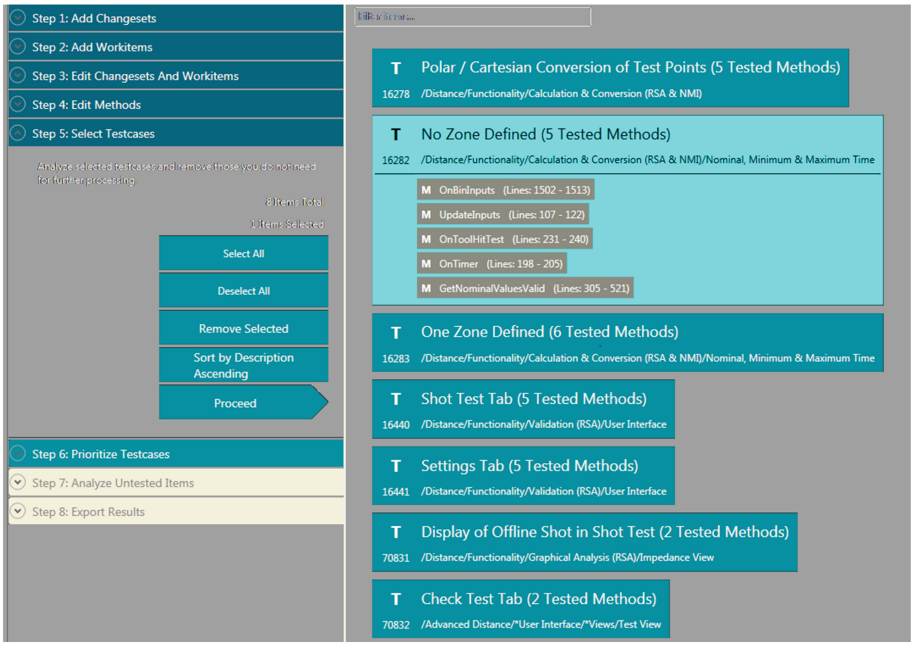
Fig. 3. Nodes and dependencies part of the basic structure of the graph database.

Client App and Export. The tester access the information stored in the graph database via a dedicated client application that guides the user through the different steps of interactively composing regression test suites. Initially the tool retrieves the test cases from the graph database that cover the investigated changes. These test cases are candidates for regression testing. The tester can sort and explore the list according to different criteria and exclude redundant or irrelevant test cases. Finally, a selected subset of the test cases is exported to the test management system where the test run is started and managed. Figure 4 depicts the Sherlock's client application displaying a list of regression test candidates to be selected.

3.2 Steps in Compiling a Regression Test Suite

Sherlock supports testers in regression testing when bug fixes and enhancements have been made to the software system. The testers use the tool for identifying test cases related to an individual change (e.g., a bug fix), for all changes within a specified date range (e.g., all fixes and enhancements combined in a maintenance release), or all changes made on a branch before it is merged back into the trunk (e.g., all changes made while implementing a new feature). The main focus of Sherlock lies in automatically proposing a list of relevant test cases that should be considered when compiling a regression test suite. The resulting test suite is the basis for subsequent regression test runs executed via the test management system. In addition, Sherlock also indicates coverage gaps, i.e., all parts of the source code that have been changed but are not covered by the selected test cases.

The following steps are typically carried out when compiling a regression test suit with Sherlock.



Step 1: Add Changesets

Step 2: Add Workitems

Step 3: Edit Changesets And Workitems

Step 4: Edit Methods

Step 5: Select Testcases

Analyze selected testcases and remove those you do not need for further processing.

8 Items Total

1 Items Selected

Step 6: Prioritize Testcases

Step 7: Analyze Untested Items

Step 8: Export Results

T Polar / Cartesian Conversion of Test Points (5 Tested Methods)
16278 /Distance/Functionality/Calculation & Conversion (RSA & NMI)

T No Zone Defined (5 Tested Methods)
16282 /Distance/Functionality/Calculation & Conversion (RSA & NMI)/Nominal, Minimum & Maximum Time

M OnBinInputs (Lines: 1502 - 1513)
M UpdateInputs (Lines: 107 - 122)
M OnToolHitTest (Lines: 231 - 240)
M OnTimer (Lines: 198 - 205)
M GetNominalValuesValid (Lines: 305 - 521)

T One Zone Defined (6 Tested Methods)
16283 /Distance/Functionality/Calculation & Conversion (RSA & NMI)/Nominal, Minimum & Maximum Time

T Shot Test Tab (5 Tested Methods)
16440 /Distance/Functionality/Validation (RSA)/User Interface

T Settings Tab (5 Tested Methods)
16441 /Distance/Functionality/Validation (RSA)/User Interface

T Display of Offline Shot in Shot Test (2 Tested Methods)
70831 /Distance/Functionality/Graphical Analysis (RSA)/Impedance View

T Check Test Tab (2 Tested Methods)
70832 /Advanced Distance/*User Interface/*Views/Test View

Fig. 4. User interface of the Sherlock client application. Tester interactively compile a regression test suite by selecting test cases and analyzing test gaps.

(1) Select Changes. The version control system of TFS keeps track of any modifications to the source code. Each modification is linked to a change set associated to a work item, e.g., a bug, a task or a requirement. Change sets group together all changes to one or several files as well as any new files created in the course of completing a work item such as resolving a bug or implementing a requirement. The tester starts with selecting the change sets for testing and/or by choosing work items, which refer to change sets relevant for testing. The selection depends on what should be tested, for example, a single bug fix, the tasks completed by a particular developer, or all changes that will be included in an upcoming release.

(2) Determine Affected Code. Based on the links between change sets and source code maintained by the TFS, the relevant code changes can be determined for the selected work items. Technically, the maintained information identifies the particular revision of a source code file in the versioning system and, thus, allows retrieving the lines that have been modified, added or removed. Using this information allows computing the list of modified methods (functions) per source code file. Resolving changes to the level of individual methods has been found to be a useful level of granularity in order to avoid large test sets [12].

(3) Query Related Tests. More than 5,000 test cases are stored in the SilkCentral test management system. The test cases are organized according to the modules of the software system. Each test case is specified by a list of steps for manual execution and a set of meta-data including, for example, the estimated execution time of the test or the test priority indicating how important the test is for testing the associated module.

In addition, optionally, coverage information is associated to each of the test cases. The coverage footprint is defined in terms of the list of files and methods executed when the test is run. For some modules the coverage information has been collected throughout full regression test runs using a coverage analysis tool. The coverage information can be used to retrieve all test cases that are able to contribute to the coverage of a changed file or method.

(4) Interactive Selection of Tests. In the previous step a list of all possibly relevant test cases is provided. Hence a test case may cover one or more changed methods, and a method may be covered by zero to many test cases. The Sherlock client allows sorting and filtering the suggested test cases according to various different criteria such as their total coverage contribution or their unique coverage contribution as well as their priority and other meta-data from the test management system. The tester can interactively select or deselect test cases and observe the impact on the overall coverage, the number of selected test cases, the overall estimated execution time, and possibly uncovered methods. Furthermore, Sherlock implements an optimization algorithm to compute test suits with minimum redundancy, which can serve as starting point for further manual adjustments.

(5) Report Test Gaps. Despite the huge number of existing test cases in the test management system, some parts of the software system are not covered by tests. For these parts, either no test cases are available or the code has recently been added and is not yet fully covered by tests. The testers are constantly extending the set of tests and aim to increase code coverage also for these parts. The goal is to close any test gaps [13] resulting from recently changed code that is lacking sufficient tests to achieve full coverage. To support the work of the testers, detected gaps are reported.

(6) Export Test Suite. Sherlock aggregates information from a broad range of sources and offers mechanisms to automatically sort and select test cases. Once the tester has made the decision which test cases to select, the selection is used to create a test suite in the test management system. There the test suite can still be adjusted before it is run. Typically, additional test cases are added, e.g., test cases that are not covering the changed code but which are mandatory or considered relevant for some reason based on the tester's experience, and the test execution order can be redefined.

4 Evaluation Design

The approach and the tool prototype have been developed in close cooperation with the company partner. Throughout development the approach and the tool support were reviewed and discussed with the testers on a regular basis. In addition, the usefulness of the tool support was assessed together with the team of testers in an *informal experiment* conducted as part of a regular workshop.

4.1 Goal and Research Questions

The goal of the evaluation was to compare *(a) regression test selection with tool support* to *(b) manual regression test selection based on personal experience* in order to explore following "research questions":

RQ1: How does the use of the tool support *impact the strategy* applied for selecting regression test cases?
RQ2: How does the use of the tool support *impact the time required* for selecting regression test cases?
RQ3: How does the use of the tool support *impact the number of selected test cases*?
RQ4: Does the use of the tool lead to a *different selection of test cases* than the manual approach?

4.2 Evaluation Setup

The evaluation of the tool support was conducted by comparing the selection of regression test cases (a) by a tester using the support of the tool Sherlock and (b) by testers manually selecting relevant test cases based on their personal knowledge and experience, i.e., the control group.

All five testers of the test team took part in the workshop. Table 1 shows the experience in number of years involved in testing at the company and the knowledge of the participants in testing the selected software system for the evaluation. *Tester A* was a senior member of the test team. He had been involved in the development of the tool and therefore performed regression test selection with Sherlock. *Tester B* was one of the most experienced testers who also had most knowledge about the modules selected for the evaluation. B's selection of test cases was considered the "gold standard" for the discussed examples. *Tester C* had medium experience and medium knowledge about the tested modules. *Tester D* was the less experienced member of the test team with only about one year. Therefore, D and *Tester E* decided to team up and to conduct the evaluation together.

Table 1. Knowledge and experience of the workshop participants.

Participant	**A**	**B**	**C**	**D+E**
Experience in years	> 10	> 10	> 5	< 1 (D) > 5 (E)
Knowledge about tested modules	some	high	some	some
Test selection approach	tool-based	manual	manual	manual

The evaluation was performed based on two major modules of the analyzed software system with an overall set of 392 specified regression test cases in the test management system. The participants were asked to select a suitable set of regression tests from this pool of test cases for different work items (bug fixes and small features) that had been resolved in the past. For each work item a textual description (e.g., initial bug report or requirement rationale, clarifying comments, details about the resolution) was available. Furthermore, the change history of each work item showed the list of

Table 2. Changes related to the example work items used in the evaluation.

Work Item	Type	Number of Check-ins	Changed Files	Changed Methods	Affected Module
WI-1	bug fix	1	1	1	M1
WI-2	bug fix	2	1	1	M1+M2
WI-3	bug fix	1	1	1	M2
WI-4	feature	5	12	28	M2
WI-5	bug fix	1	5	8	M2
WI-6	bug fix	1	1	2	M1

changes the developers made to the source code. We prepared a list of eight work items for the evaluation. Two work items were discarded in the workshop as not directly related to the two selected modules. Table 2 shows the six remaining work items that were analyzed and discussed further.

4.3 Limitations and Validity Threats

The blueprint of the evaluation is similar to an experiment design. However, it has not been conducted in a controlled environment since it was embedded in a workshop with the goal to discuss usage scenarios and envisioned benefits of the tool support.

The investigated usage scenario and the selected examples are specific for the project and company context, and they were deliberately chosen in order to foster the discussion with the members of the test team. The evaluation has been performed in an informal, interactive setting that promoted feedback and new insights rather than producing generalizable results. The participants were eager to produce accurate and representative results. However, the workshop setting also led to occasional disruptions such as participants leaving the room or taking phone calls.

We addressed these limitations by documenting all workshop activities, discussions as well as interruptions in a detailed protocol and by conducting an ex-post analysis of the results with one of the participants to confirm our observations before drawing conclusions. Due to the small number of analyzed cases and participants the findings were not tested for statistical significance.

5 Results and Discussion

In the workshop the testers (*A*, *B*, *C* and *D* + *E*) were given the task to select regression test cases for each of the previously chosen work items (*WI-1*, *WI-2* etc.). Each work item was processed as follows. First, the unique ID of the work items was announced and the starting time was recorded. The testers looked up the work item in TFS, read the work item description, and selected the test cases they considered relevant for regression testing. They specified their selection in a predefined Excel sheet by listing the test case name and ID of the test cases specified in the SilkCentral test managment system. Finally they sent their list by email to the workshop moderator. The time stamp of the email was defined as completion time. Once all testers had sent their selections, the individual results were presented side by side and discussed by the whole team. Questions that were typically asked included, for example, why a particular test case was selected or not selected by a tester, whether a particular test case could be considered equivalent to another with respect to a certain test objective, or what additional test cases may have to be designed. We asked further questions about how the testers performed test case selection, what information source they consulted, etc. and documented the discussion and feedback, which was used to discuss and answer our research questions.

5.1 Strategies Used for Test Case Selection (RQ1)

The first research question investigates the way the testers perform the task of regression test selection with or without tool support. In both cases the testers have to make decisions that require experience and knowledge. The rationale of this question is to explore which kinds of experience and knowledge are required when different strategies for test case selection are applied.

(a) Regression test selection using tool support. When using Sherlock, tester *A* studied the description of the work item and queried the associated changes stored in the Sherlock database. The tool presented a list of all candidate test cases covering the changed methods from which the final regression test suite was derived by deselecting the tests considered redundant or irrelevant. The decision which test cases to select or deselect was mainly based on the short description of the test cases shown in Sherlock. Tester *A* did not add any test cases to the regression test suite other than those initially proposed by the tool.

(b) Manual regression test selection. The testers *B*, *C*, and *D* + *E* performing manual selection also started by studying the description of the work item. Then they browsed the hierarchically structured set of existing test cases in the test management system and built up their regression test suite by adding the relevant tests to the initially empty list. The most experienced tester *B* occasionally also looked at the list of changed files associated to the work items and usually selected individual test cases based on his detailed knowledge of the modules. In contrast, the other testers tended to selected all test cases related to a specific function (e.g., all test cases concerned with "Reporting"). For selecting a specific test case out of all related test cases the less experienced tester usually had to check the detailed test case description and the list of specified test steps.

Our observations and the feedback from the testers show that **the tool support leads to a test case selection strategy (*initial proposal and deselection*) that is different from the manual selection approach (*bottom-up selection*).**

The discussion of the effect the two different strategies have on the produced regression test suites constitutes the answers to the following research questions.

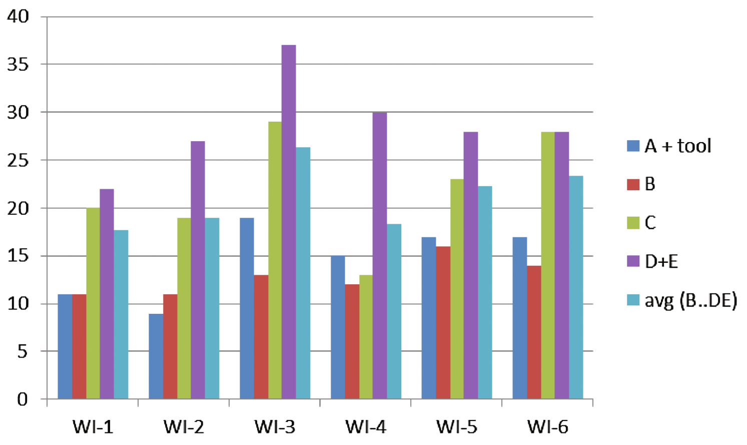


Fig. 5. Time required for regression test selection by work item and tester. (Color figure online)

5.2 Time Required for Regression Test Selection (RQ2)

The second research question investigates if the tool support impacts the time that is necessary for selecting regression test cases. The purpose of this question lies in identifying possible usability issues related to the handling of the tool, which may result in an observable holdup when Sherlock is used.

Figure 5 shows the time required (by testers, in minutes) to select the regression tests for each of the work items. The first bar (dark blue) is the time of tester *A* using the tool Sherlock. For all work items, tester *A* required less time than the *average of the other testers* (last bar, shown in light blue) selecting the regression tests manually. In most cases tester *A* performed the test selection in about the same time as the most experienced tester *B* (second bar, shown in red).

The total of test case selection for all 6 work items, the time required by tester *A* was 1:28 h, the minimum overall time was 1:17 h (tester *B*), the maximum time was 2:52 h (team of testers *D* + *E*), and the average overall time of the testers performing manual selection was 2:07 h.

The results indicate that **the usage of the tool does not have a noticeable adverse effect on the time required for selecting regression test cases.**

5.3 Number of Selected Test Cases (RQ3)

The third research question aims to investigate how the tool support impacts the number of test cases selected for regression testing. The actual number of selected test cases is likely depending on the tester's personal experience and the knowledge he or she has about the tested modules. Therefore, we do not expect that the number of test cases selected by the tester using Sherlock differs much from those of the other testers performing manual test case selection. However, an obvious indicator that the tool support actually impacts the selection would be if Sherlock constraints the number of initially proposed tests too much and, thus, the tester is not able to select the appropriate test cases.

Table 3 provides an overview of the number of test cases selected by the testers. The two columns on the left of the table labeled *Tool* and *A + Tool* show the number of test cases initially proposed by the tool Sherlock and the number of test cases finally selected by tester *A*. Sherlock proposed 146.8 test cases on average, which were reduced to an average of 7.3 tests (about 5 %) by tester *A*. The right column (*Avg BCDE*) shows that the average number of test cases selected by the testers who performed a manual selection is 13.3. On average, thus, they manually selected about twice as many tests as tester *A*.

Table 3. Number of test cases selected per tester and average.

Work Item	**Tool**	**A+Tool**	**B**	**C**	**D+E**	**Avg BCDE**
WI-1	14	9	5	6	16	9.0
WI-2	48	3	1	11	14	8.7
WI-3	200	4	10	16	19	15.0
WI-4	222	15	30	-	34	32.0
WI-5	222	6	6	5	2	4.3
WI-6	175	7	1	15	17	11.0
Average	146.8	7.3	8.8	10.6	17.0	13.3

Moreover, tester *A* using the tool support selected roughly the same number of test cases as tester *B*. In those cases where tester *A* selected less tests than tester *B* (*WI-3* and *WI-4*), the reason is obviously not a confined preselection by Sherlock. For four of the work items (including *WI-3* and *WI-4*) Sherlock found a very large number of test cases since all of them covered one or more of the changed methods. For these cases Sherlock was not able to make an appropriate preselection and, thus, almost all test cases for the particular module were returned.

It can be observed from these results that **the tool support did not lead to a specifically high or low number of selected test cases.**

5.4 Differences in Selected Test Cases (RQ4)

The fourth research question investigates if the use of the tool produces a different set of regression test cases than the manual selection approach. The purpose of this question is to identify the need for a further analysis of the applied selection strategies and the resulting test cases in order to adjust and improve the tool support.

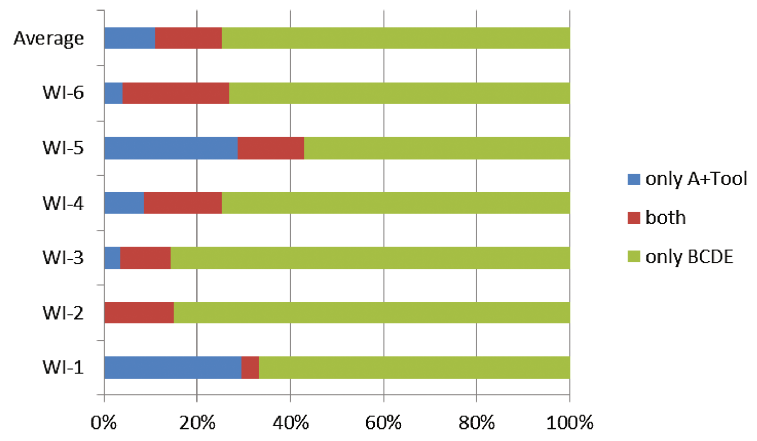


Fig. 6. Test cases selected uniquely by tester *A*, by the group of the testers *BCDE*, or by both.

Figure 6 shows the number of test cases that were only selected by tester *A* using Sherlock (blue portion), only by the group of testers performing manual selection (testers *BCDE*, green portion), or by both (red portion). The top bar shows the average shares: 11 % of the test cases were uniquely selected by tester *A* and 75 % uniquely by the testers *BCDE*. 14 % of the test cases were selected by both.

On the first glance the agreement in terms of the 14 % test cases shared by tester *A* and the testers *BCDE* seems small. However, the relatively low share is not surprising since tester *A* generally selected a low number of test cases, 44 in total for all six work items (or 7.3 tests on average as shown above). In contrast, *BCDE* together selected 155 unique test cases in total.

We also compared the selection made by tester *A* with tool support to the “gold standard” provided by tester *B*. The intersection $A \cap B$ shows a 23 % overlap between the test cases selected by *A* and those selected by *B*. In the follow-up discussion the testers explained the differences by the presence of redundant test cases of which only one or a few are selected. Since these tests are considered equivalent, it is merely a personal choice which test case out of the several redundant ones is eventually selected by a tester.

The regression test suited produced with the tool support differs from the regression test suited produced manually to about the same extent as the test suites produced with manual selection by different testers.

Finally, when computing the agreement between the testers *B*, *C* and $D + E$ in terms of the intersection $B \cap C \cap DE$ one can observe that on average only 6 % of the manually selected test cases were selected by all three of them. The relatively small overlap between the tester *A* and the group of testers *BCDE* can therefore not be attributed to the tool support.

6 Conclusions and Future Work

In this paper we described a tool-based approach for regression test selection. The tool Sherlock had been implemented as an aid for testers to compile a regression test suite with the goal to cover all changed files and methods. Sherlock incorporates information from different data sources such as the version control system used by the developers, the test management system containing the available test cases, and the code coverage footprint of the test cases that is used to link the individual test cases to the changed source code.

Sherlock presents a list of all candidate test cases covering the changed files and methods. The regression test suite is derived by deselecting the test cases considered redundant or irrelevant. In contrast, the testers performing manual selection typically browse the set of existing test cases in the test management system and built up their regression test suite by adding the relevant tests to the initially empty suite.

An evaluation conducted together with the members of the test team showed that the tester who used Sherlock was able produce test suites with less or equal effort and at the same level of accuracy as the testers in the control group who selected the test cases manually. The performance of the tester using Sherlock was about the same as the one showed by a highly experienced tester who had detailed knowledge about the changes to be tested.

For junior testers, who lack the necessary background and experience, Sherlock can be considered a valuable aid providing guidance in selecting appropriate regression test cases. Experienced testers find the tool support useful to double-check and enhance a manually compiled test suite with the automatically proposed list of tests. Furthermore, the tool also allows identifying coverage gaps in the set of available test cases.

In developing the tool support for change-based regression testing we encountered several challenges and open issues, which we consider of general importance when establishing test case selection in practice. The key lessons we derived from these challenges and issues are also the topics that drive our future work.

Lack of details provided by version control systems. Modern tools for task management and version control are able link work items such as bug fixes and features to changes in the source code shown in form of the list of changed, added and deleted lines in the affected source files. The source files are treated as standard text files. Their internal structure defined by the implementation is ignored by the tools. However, the semantic of the change is different whether a changed line maps to a class, method, comment, blank line, etc. Restoring this information is a complex task that requires parsing the changed files and, thus, dealing with different programming languages and technologies.

Selecting from redundant test cases. The code coverage footprint of a test case reveals what parts of the code are executed by the particular test. When two different test cases have the exactly same coverage footprint they may be considered redundant in change-based regression testing. However, these tests may still differ in the way they assert the results and in what properties and aspects of the system's functionality they verify. For example, one test checks only the results shown on the user interface while the other also verifies the values stored in the database. Code coverage information alone is not sufficient to determine the relevant test cases for regression testing. In Sherlock, thus, the tester currently has to decide which test cases to select in case there are several "redundant" tests proposed for a change.

Large and polluted coverage footprints. The execution of a test case comprises many different steps such as starting the tested module, logging in as a test user, opening network connections and files to setup the test scenario, exercising the functionality under test, and checking the outcome and possible side-effects. Therefore, many different parts of the system are exercised throughout test execution, including parts that are not actually subject to testing. Nevertheless, these parts may still be included in coverage measurement. In consequence they pollute the coverage footprint of the test and inflate it with irrelevant entries. Test cases with such coverage footprints often show up as false positives in the initially list of test cases proposed by the tool support. Furthermore, classes and methods required for the module startup or which are frequently used in the test setup are included in the coverage footprint of many test cases. If a change set contains such a method, a huge set of test cases is proposed. Sherlock deals with this problem by excluding coverage information collected in the setup phase and by considering the hit count in determining the relevant tests, i.e., the number of times a method or class has been executed in testing.

Creating and maintaining coverage footprints. The availability of up-to-date coverage footprints is essential for the described change-based regression testing approach [14]. Collecting these footprints from test execution is often the only way to obtain the information necessary for establishing the connection between changes and tests. In the studied project, similar information could not be retrieved via static analysis due to the size of the software system and the various different technologies used in its implementation. For a project that already has a large base of test cases it is a major challenge to establish the critical mass of coverage information that is necessary to make the tool support useful for the testers as part of their daily work. The testers will only start using the tool if they trust that the information it provides is accurate and complete. Furthermore, keeping code coverage information up-to-date requires frequent re-execution of the tests, which is particularly costly for manual test cases. Organizational measures are necessary to keep the effort required for maintaining the coverage footprints in balance with the benefits of a change-based regression testing approach.

Acknowledgments. The research reported in this paper has been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH. Furthermore, the authors also thank the industry partner and the members of test team for their valuable time, support and feedback.

References

1. Lehman, M.M., Belady, L.A. (eds.): Program Evolution: Processes of Software Change. Academic Press Prof., London (1985)
2. Ammann, P., Offutt, J.: Introduction to Software Testing, 1st edn. Cambridge University Press, Cambridge (2008)
3. Juergens, E., Hummel, B., Deissenboeck, F., Feilkas, M., Schlogel C., Wubbeke, A.: Regression test selection of manual system tests in practice. In: 15th European Conference on Software Maintenance and Reengineering (CSMR), pp. 309–312 (2011)
4. Rothermel, G., Harrold, M.J.: Analyzing regression test selection techniques. IEEE Trans. Softw. Eng. **22**(8), 529–551 (1996)
5. Engström, E., Runeson, P., Skoglund, M.: A systematic review on regression test selection techniques. Inf. Softw. Technol. **52**(1), 14–30 (2010)
6. Yoo, S., Harmann, M.: Regression testing minimisation, selection, and prioritisation: a survey. Softw. Test Verif. Reliab. **22**(2), 67–120 (2012)
7. Gligoric, M., Eloussi, L., Marinov, D.: Practical regression test selection with dynamic file dependencies. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015), pp. 211–222 (2015)
8. Elbaum, S., Rothermel, G., Penix, J.: Techniques for improving regression testing in continuous integration development environments. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014), pp. 235–245 (2014)
9. Li, B., Sun, X., Leung, H., Zhang, S.: A survey of code-based change impact analysis techniques. Softw. Test Verif. Reliab. **23**(8), 613–646 (2012)
10. Rothermel, G., Harrold, M.J.: A safe, efficient regression test selection technique. ACM Trans. Softw. Eng. Methodol. **6**(2), 173–210 (1997)
11. Chen, Y.-F., Rosenblum, D.S., Vo, K.-P.: TestTube: a system for selective regression testing. In: Proceedings of the 16th International Conference on Software Engineering (ICSE 1994) (1994)
12. Buchgeher, G., Ernstbrunner, C., Ramler, R., Lusser, M.: Towards tool-support for test case selection in manual regression testing. In: Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 74–79 (2013)
13. Eder, S., Hauptmann, B., Junker, M., Juergens, E., Vaas, R., Prommer, K.H.: Did we test our changes? Assessing alignment between tests and development in practice. In: Proceedings of the 8th International Workshop on Automation of Software Test (AST), pp. 107–110 (2013)
14. Beszedes, A., Gergely, T., Schrettner, L., Jasz, J., Lango, L., Gyimothy, T.: Code coverage-based regression test selection and prioritization in WebKit. In: Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), pp. 46–55 (2012)